CLAIMS

1. A computer system that employs a plurality of threads of execution to perform a parallel-execution operation in which the threads identify tasks dynamically and in which the computer system:

A) provides a global status word that includes a separate status-word field associated with each of the threads; and

B) so operates the threads that each thread:

i) executes a task-finding routine to find tasks previously identified dynamically and performs tasks thereby found, with the status-word field associated with that thread containing an activity-representing value, until the task-finding routine finds no more tasks;

ii) when the task-finding routine finds no more tasks, sets the contents of the status-word field associated with that thread to an inactivity-indicating value;

iii) while the status-word field associated with any other thread contains an activity-indicating value, searches for a task and, if it finds one, sets the status-word field to the activity-indicating value before attempting to execute a task; and

iv) if none of the status-word fields contains an activity-indicating value, terminates its performance of the parallel-execution operation.

2. A computer system as defined in claim 1 wherein the parallel-execution operation is a garbage-collection operation.

3. A computer system as defined in claim 1 wherein:

A) each thread has associated with it a respective work queue in which it places task identifiers of tasks that identifies dynamically;

B) the task-finding routine executed by an executing thread includes performing an initial search for a task identifiers in the work queue associated with the executing thread and, if that work queue contains no task identifiers that the executing thread can claim, thereafter performing a further search for a task identifier in at least one other task-storage location.

4. A computer system as defined in claim 3 wherein the parallel-execution operation is a garbage-collection operation.

5. A computer system as defined in claim 3 wherein the at least one other task-storage location includes at least one work queue associated with a thread other than the executing thread.

6. A computer system as defined in claim 5 wherein:

A) there is a size limit associated with each work queue;

B) when a given thread dynamically identifies a given task that would cause the number of task entries in the work queue associated with the given thread to exceed the size limit if a task identifier that identifies it were placed in that work queue, the given thread instead places that task identifier in an overflow list instead of in that work queue; and

C) the at least one other task-storage location includes at least one such overflow list.

7. A computer system as defined in claim 5 wherein the task-finding routine includes selecting in a random manner the at least one work queue associated with a thread other than the executing thread.

8. A computer system as defined in claim 5 wherein the further search includes repeatedly searching a work queue associated with a thread other than the executing thread until the executing thread thereby finds a task or has performed a number of repetitions equal to a repetition limit greater than one.

9. A computer system as defined in claim 8 wherein the task-finding routine includes selecting in a random manner the at least one work queue associated with a thread other than the executing thread.

10. A computer system as defined in claim 3 wherein:

A) there is a size limit associated with each work queue;

B) when a given thread dynamically identifies a given task that would cause the number of task entries in the work queue associated with the given thread to exceed the size limit if a task identifier that identifies it were placed in that work queue, the given thread instead places that task identifier in an overflow list instead of in that work queue; and

C) the at least one other task-storage location includes at least one such overflow list.

11. A computer system as defined in claim 1 wherein the status word fits in a memory location accessible in a single machine instruction.

12. A computer system as defined in claim 11 wherein the parallel-execution operation is a garbage-collection operation.

13. A computer system as defined in claim 11 wherein each status-word field is a single-bit field.

14. A computer system as defined in claim 13 wherein the activity-indicating value is a logic one and the inactivity-indicating value is a logic zero.

15. For employing a plurality of threads of execution to perform a parallel-execution operation in which the threads identify tasks dynamically, a method comprising:

A) providing a global status word that includes a separate status-word field associated with each of the threads; and

B) so operating the threads that each thread:

i) executes a task-finding routine to find tasks previously identified dynamically and performs tasks thereby found, with the status-word field associated with that thread containing an activity-representing value, until the task-finding routine finds no more tasks;

ii) when the task-finding routine finds no more tasks, sets the contents of the status-word field associated with that thread to an inactivity-indicating value;

iii) while the status-word field associated with any other thread contains an activity-indicating value, searches for a task and, if it finds one, sets the status-word field to the activity-indicating value before attempting to execute a task; and

iv) if none of the status-word fields contains an activity-indicating value, terminates its performance of the parallel-execution operation.

16. A method as defined in claim 15 wherein the parallel-execution operation is a garbage-collection operation.

17. A method as defined in claim 15 wherein:

A) each thread has associated with it a respective work queue in which it places task identifiers of tasks that identifies dynamically;

B) the task-finding routine executed by an executing thread includes performing an initial search for a task identifiers in the work queue associated with the executing thread and, if that work queue contains no task identifiers that the executing thread can claim, thereafter performing a further search for a task identifier in at least one other task-storage location.

18. A method as defined in claim 17 wherein the parallel-execution operation is a garbage-collection operation.

19. A method as defined in claim 17 wherein the at least one other task-storage location includes at least one work queue associated with a thread other than the executing thread.

20. A method as defined in claim 19 wherein:

A) there is a size limit associated with each work queue;

B) when a given thread dynamically identifies a given task that would cause the number of task entries in the work queue associated with the given thread to exceed the size limit if a task identifier that identifies it were placed in that work queue, the given thread instead places that task identifier in an overflow list instead of in that work queue; and

C) the at least one other task-storage location includes at least one such overflow list.

21. A method as defined in claim 19 wherein the task-finding routine includes selecting in a random manner the at least one work queue associated with a thread other than the executing thread.

22. A method as defined in claim 19 wherein the further search includes repeatedly searching a work queue associated with a thread other than the executing thread until the executing thread thereby finds a task or has performed a number of repetitions equal to a repetition limit greater than one.

23. A method as defined in claim 22 wherein the task-finding routine includes selecting in a random manner the at least one work queue associated with a thread other than the executing thread.

24. A method as defined in claim 17 wherein:

A) there is a size limit associated with each work queue;

B) when a given thread dynamically identifies a given task that would cause the number of task entries in the work queue associated with the given thread to exceed the size limit if a task identifier that identifies it were placed in that work queue, the given thread instead places that task identifier in an overflow list instead of in that work queue; and

C) the at least one other task-storage location includes at least one such overflow list.

25. A method as defined in claim 15 wherein the status word fits in a memory location accessible in a single machine instruction.

26. A method as defined in claim 25 wherein the parallel-execution operation is a garbage-collection operation.

27. A method as defined in claim 25 wherein each status-word field is a single-bit field.

28. A method as defined in claim 27 wherein the activity-indicating value is a logic one and the inactivity-indicating value is a logic zero.

29. A storage medium containing instructions readable by a computer system to configure the computer system to employ a plurality of threads of execution to perform a parallel-execution operation in which the threads identify tasks dynamically and in which the computer system:

A) provides a global status word that includes a separate status-word field associated with each of the threads; and

B) so operates the threads that each thread:

i) executes a task-finding routine to find tasks previously identified dynamically and performs tasks thereby found, with the status-word field associated with that thread containing an activity-

representing value, until the task-finding routine finds no more tasks;

ii) when the task-finding routine finds no more tasks, sets the contents of the status-word field associated with that thread to an inactivity-indicating value;

iii) while the status-word field associated with any other thread contains an activity-indicating value, searches for a task and, if it finds one, sets the status-word field to the activity-indicating value before attempting to execute a task; and

iv) if none of the status-word fields contains an activity-indicating value, terminates its performance of the parallel-execution operation.

30. A storage medium as defined in claim 29 wherein the parallel-execution operation is a garbage-collection operation.

31. A storage medium as defined in claim 29 wherein:

A) each thread has associated with it a respective work queue in which it places task identifiers of tasks that identifies dynamically;

B) the task-finding routine executed by an executing thread includes performing an initial search for a task identifiers in the work queue associated with the executing thread and, if that work queue contains no task identifiers that the executing thread can claim, thereafter performing a further search for a task identifier in at least one other task-storage location.

32. A storage medium as defined in claim 31 wherein the parallel-execution operation is a garbage-collection operation.

33. A storage medium as defined in claim 31 wherein the at least one other task-storage location includes at least one work queue associated with a thread other than the executing thread.

34. A storage medium as defined in claim 33 wherein:

A) there is a size limit associated with each work queue;

B) when a given thread dynamically identifies a given task that would cause the number of task entries in the work queue associated with the given thread to exceed the size limit if a task identifier that identifies it were placed in that work queue, the given thread instead places that task identifier in an overflow list instead of in that work queue; and

C) the at least one other task-storage location includes at least one such overflow list.

35. A storage medium as defined in claim 33 wherein the task-finding routine includes selecting in a random manner the at least one work queue associated with a thread other than the executing thread.

36. A storage medium as defined in claim 33 wherein the further search includes repeatedly searching a work queue associated with a thread other than the executing thread until the executing thread thereby finds a task or has performed a number of repetitions equal to a repetition limit greater than one.

37. A storage medium as defined in claim 36 wherein the task-finding routine includes selecting in a random manner the at least one work queue associated with a thread other than the executing thread.

38. A storage medium as defined in claim 31 wherein:

A) there is a size limit associated with each work queue;

B) when a given thread dynamically identifies a given task that would cause the number of task entries in the work queue associated with the given thread to exceed the size limit if a task identifier that identifies it were placed in that work queue, the given thread instead places that task identifier in an overflow list instead of in that work queue; and

C) the at least one other task-storage location includes at least one such overflow list.

39. A storage medium as defined in claim 29 wherein the status word fits in a memory location accessible in a single machine instruction.

40. A storage medium as defined in claim 39 wherein the parallel-execution operation is a garbage-collection operation.

41. A storage medium as defined in claim 39 wherein each status-word field is a single-bit field.

42. A storage medium as defined in claim 41 wherein the activity-indicating value is a logic one and the inactivity-indicating value is a logic zero.

43. A computer signal representing a sequence of instructions that, when executed by a computer system, configures the computer system to employ a plurality of threads of execution to perform a parallel-execution operation in which the threads identify tasks dynamically and in which the computer system:

A) provides a global status word that includes a separate status-word field associated with each of the threads; and

B) so operates the threads that each thread:

i) executes a task-finding routine to find tasks previously identified dynamically and performs tasks thereby found, with the status-word field associated with that thread containing an activity-representing value, until the task-finding routine finds no more tasks;

ii) when the task-finding routine finds no more tasks, sets the contents of the status-word field associated with that thread to an inactivity-indicating value;

iii) while the status-word field associated with any other thread contains an activity-indicating value, searches for a task and, if it finds one, sets the status-word field to the activity-indicating value before attempting to execute a task; and

iv) if none of the status-word fields contains an activity-indicating value, terminates its performance of the parallel-execution operation.

44. A computer signal as defined in claim 43 wherein the parallel-execution operation is a garbage-collection operation.

45. A computer signal as defined in claim 43 wherein:

A) each thread has associated with it a respective work queue in which it places task identifiers of tasks that identifies dynamically;

B) the task-finding routine executed by an executing thread includes performing an initial search for a task identifiers in the work queue associated with the executing thread and, if that work queue contains no task identifiers that the executing thread can claim, thereafter performing a further search for a task identifier in at least one other task-storage location.

46. A computer signal as defined in claim 45 wherein the parallel-execution operation is a garbage-collection operation.

47. A computer signal as defined in claim 45 wherein the at least one other task-storage location includes at least one work queue associated with a thread other than the executing thread.

48. A computer signal as defined in claim 47 wherein:

A) there is a size limit associated with each work queue;

B) when a given thread dynamically identifies a given task that would cause the number of task entries in the work queue associated with the given

thread to exceed the size limit if a task identifier that identifies it were placed in that work queue, the given thread instead places that task identifier in an overflow list instead of in that work queue; and

C) the at least one other task-storage location includes at least one such overflow list.

49. A computer signal as defined in claim 47 wherein the task-finding routine includes selecting in a random manner the at least one work queue associated with a thread other than the executing thread.

50. A computer signal as defined in claim 47 wherein the further search includes repeatedly searching a work queue associated with a thread other than the executing thread until the executing thread thereby finds a task or has performed a number of repetitions equal to a repetition limit greater than one.

51. A computer signal as defined in claim 50 wherein the task-finding routine includes selecting in a random manner the at least one work queue associated with a thread other than the executing thread.

52. A computer signal as defined in claim 45 wherein:

A) there is a size limit associated with each work queue;

B) when a given thread dynamically identifies a given task that would cause the number of task entries in the work queue associated with the given thread to exceed the size limit if a task identifier that identifies it were placed in that work queue, the given thread instead places that task identifier in an overflow list instead of in that work queue; and

C) the at least one other task-storage location includes at least one such overflow list.

53. A computer signal as defined in claim 43 wherein the status word fits in a memory location accessible in a single machine instruction.

54. A computer signal as defined in claim 53 wherein the parallel-execution operation is a garbage-collection operation.

55. A computer signal as defined in claim 53 wherein each status-word field is a single-bit field.

56. A computer signal as defined in claim 55 wherein the activity-indicating value is a logic one and the inactivity-indicating value is a logic zero.

57. A computer system that employs a plurality of threads of execution to perform a parallel-execution operation in which the threads identify tasks dynamically, the computer system including:

A) means for providing a global status word that includes a separate status-word field associated with each of the threads; and

B) means for so operating the threads that each thread:

i) executes a task-finding routine to find tasks previously identified dynamically and performs tasks thereby found, with the status-word field associated with that thread containing an activity-representing value, until the task-finding routine finds no more tasks;

ii) when the task-finding routine finds no more tasks, sets the contents of the status-word field associated with that thread to an inactivity-indicating value;

iii) while the status-word field associated with any other thread contains an activity-indicating value, searches for a task and, if it finds one, sets the status-word field to the activity-indicating value before attempting to execute a task; and

iv) if none of the status-word fields contains an activity-indicating value, terminates its performance of the parallel-execution operation.